Australasian Journal of Educational Technology

2011, 27(6), 896-906

A flexible, extensible online testing system for mathematics

Tim Passmore, Leigh Brookshaw and Harry Butler
University of Southern Queensland

An online testing system developed for entry-skills testing of first-year university students in algebra and calculus is described. The system combines the open-source computer algebra system *Maxima* with computer scripts to parse student answers, which are entered using standard mathematical notation and conventions. The answers can involve structures such as: lists, variable-precision-floating-point numbers, integers and algebra. This flexibility allows more sophisticated testing designs than the multiple choice, or exact match, paradigms common in other systems, and the implementation is entirely vendor neutral. Experience using the system and ideas for further development are discussed.

Introduction

Computer-aided assessment (CAA) has been widely adopted in higher education and many systems have been developed which have overlapping functionality. For example, Dopper and Sjoer (2004) report that seventeen different systems have been built over fifteen years at Delft University of Technology alone. Our experience at the University of Southern Queensland has involved several systems over a similar period. CAA systems are becoming more sophisticated in terms of the variety of input they will accept, for example, scanning semantic similarities in student C programs (Wang, Su, Wang & Ma, 2007), or grading text of open-ended problem solutions in earth science (Wang, Chang & Li, 2008). Over a similar period, mathematics educators have been evaluating computer algebra systems (CAS) as tools in the learning of mathematics (Artigue, 2002; Nicaud, Bouhineau & Chaachoua, 2004; Klasa, 2010; Berger, 2010). In the assessment of mathematics, in particular, the use of computer algebra systems (CAS) as part of CAA has become commonplace (Sangwin, 2006) and it has been predicted that all CAA will eventually link to CAS (Sangwin, 2003). Computer algebra offers the capacity to accept student algebraic input directly and either evaluate its equivalence to a known answer, or manipulate the input further to determine its properties. This allows the question author to avoid the shortcomings of multiple choice formats where the student knows that one of the given distracters is correct. For example, solving an equation is significantly different from verifying whether one of a number of alternatives is a solution; the student can avoid answering the question as set by checking each answer by substitution.

The first system to make use of CAS in this way was *AiM* (Klai, Kolokolnikov & Van den Bergh, 2000) and its descendant *AiM-TtH* (Strickland, 2002; Sangwin, 2003) which used the commercial computer algebra system *Maple*, but it required students to know *Maple* syntax and its question authoring system required programming expertise (Sangwin & Grove, 2006). *Maple* has its own proprietary system, *MapleTA*, while other

systems have been developed for various CAS: *CalMath* uses *Mathematica*, *CABLE* (Naismith & Sangwin, 2004) uses *Axiom* and *STACK* (Sangwin & Grove, 2006) which uses *Maxima*. Each of these systems provided the student with an interactive mathematical experience within a virtual learning environment, that is they were tutorial systems as well as testing systems. Tutorial assessment systems have been used to prepare students for standardised testing (Feng, Heffernan & Koedinger, 2009) and in remedial teaching (Wang, 2010).

At the University of Southern Queensland we wanted to use CAA for entry-level testing (Engelbrecht & Harding, 2004) — assessing the core prerequisite, entry-level skills of students undertaking first-year algebra and calculus. First-year cohorts were large in number, up to 900 students, making manual testing and marking impractical. A wide variation in the skill levels of commencing students had been reported (Jourdan, Cretchley & Passmore, 2007) and early detection and intervention was desirable for the weaker students. It was essential that the entry-level testing be done early in the semester, with minimal expense to the student; that immediate feedback be available to identify those who may be at risk without early remediation; and that the test be delivered online, to make it available to off campus students (i.e. distance students). An existing CAA system was deemed unsuitable because it only allowed questions which were either multiple choice, or exact character-match to a given answer. It was also slow to use as LaTeX-coded mathematical objects were first rendered as bitmap images that were subsequently re-embedded into HTML to generate the on screen questions. The quality of these images was thus hardware dependent, and often quite poor. A better system was needed, and we developed the *Online Testing System* (OTS) described below.

Past experience indicated that the most likely areas of difficulty were algebra, functions, trigonometry and inequalities, which suggested immediately the use of a CAS. However, we did not want to burden students with learning any specialised input syntax so early in their university experience, opting instead for an "informal syntax", along the lines proposed by Sangwin and Ramsden (2007), which was as close as possible on a keyboard to standard mathematical notation and convention. Lastly, we had no budget for proprietary software or licence fees, so we needed a vendor neutral system which used existing open source software.

Features of the Online Testing System

Over the past few years, we have been using an OTS which combines the open source CAS *Maxima* with PHP scripts and XML configuration files to parse, evaluate and mark student answers which are submitted online. Since it employs standard web technology, the system is accessible to students with only minimal resources, namely a web browser (Java is not required) and a PDF document viewer. It is a testing system rather than a tutorial system and has a number of features which address the difficulties outlined earlier.

Separate question delivery and answer collection

The OTS separates question delivery from answer collection. Students must login to download the question paper as a PDF document. They can either print this or view it on screen; since it is scalable it can be viewed at any desired resolution (even on mobile devices), without loss of readability. Students can work on the paper, away from the

computer if that is the way they prefer, before logging in again to submit their answers via a standard web form. Access to the online submission form is unlimited, until the designated closing date, which allows distance students in different time zones to work when it suits them.

Another advantage of PDF question delivery is that it makes the OTS independent of the word processing or authoring system used by question authors. No programming expertise is required and the questions are displayed exactly as the author intended.

Disadvantages are that students cannot work interactively and check their syntax as with *AiM* and *STACK*. Also the question paper is printable and therefore not secure, though it would be possible to encrypt the PDF so that it was not printable. For our particular application to entry-level testing, this was not considered a serious problem as the intention is to eventually build up question banks and randomly generate question papers. More attention would need to be given to these issues if the OTS were applied for more security-sensitive purposes such as examinations.

Two-stage Check and Submit

When students enter their answers there are two stages to completing submission. The first stage is to select a *Check* button, after which any blank fields are highlighted and any specific reminder information is fed back for each question. Reminders are specified by the question author, some examples are:

- You should supply a negative answer.
- A simple solution of the form x/y is expected. If you don't supply a simple solution it will be marked incorrect.
- Give numerical values only, accurate to 4 decimal places. Do *not* give angles in degrees. Separate multiple answers with a comma; the order of the answers is unimportant.

No syntax checking is performed at this stage, though this would be a desirable enhancement in future. The reminders are intended to help the students check their work before final submission. Students can change their answers at this stage, if they wish, by using the *Back* button in their browser. They can check and revise as many times as they like, though they only get the same reminders each time.

When checking is complete, the student uploads his or her answers by selecting the *Submit* button on the form. Their answers are evaluated, marked, the marks totalled and returned to the student's screen with specific feedback. The submitted answers and resulting marks are stored for future processing (e.g. uploading to central grading database) by the OTS. Question authors can optionally specify two types of feedback, one if the question is correct and the other if the question is incorrect. This functionality may be extended in the future.

Informal input syntax

As stated earlier, we did not require students to learn the syntax for *Maxima*, or commercial CAS systems like *Maple* or *Mathematica*, before sitting the test. Instead, we wanted them to be able to input their answers using an informal linear syntax, as natural as calculator entry, and as close to standard mathematical notation and

convention as is possible on a keyboard. The implementation should be browser independent and allow for that fact that weaker students are often careless with syntax. For example, (2x-1)(x+3) could be entered either as (2x-1)(x+3), or (x+3)(2x-1), or as (2*x-1)*(x+3), the latter being the full *Maxima* syntax. If the student entered either of the first two variants, they are prefiltered using a PHP script to "add stars" before being passed to the CAS *Maxima*, which expands the brackets before checking the answer, thus making the order of the factors irrelevant. Usually, student input was not passed directly to *Maxima*, but was prefiltered and parsed by one or more PHP scripts. These scripts have been tested to ensure that they do not add any errors to student input, but the raw input is also captured in case of any doubt. Thus, the raw input can be reviewed if the student feels that a mistake has been made.

Sometimes, it is not necessary to call *Maxima* at all, resulting in faster processing performance. Consider the question:

Express $\dfrac{16\left(a^2b^4\right)^{-\frac{1}{2}}}{b^{-3}}$ as a simple fraction involving no negative powers.

Student input was parsed using XML code as shown in the Appendix Example 1 (AE1).

The student input is matched to a regular expression test (regexp) which is handled directly by PHP. But first the string is prefiltered to remove leading, trailing and embedded white space, convert alpha characters to lower case, remove any explicit multiplication symbols *, substitute any grouping done with [ ] or { } pairs by ( ) and finally, remove all ( ) pairs. The filtered string is then compared to the regexp between the @ characters in the <value> element (see AE1). In this case, the @ character functions as a regexp delimiter, but more generally delimits any string to be passed to another script (@ was chosen as it is unlikely to appear in student input). The correct answer 16b/a is matched if the filtered string is either 16b/a or b16/a, the second form being necessary if the student entered the correct string b*16/a.

A second example is the following question.

Expand (x+1)(-2x+1)(x-3)

(Exponents or powers must be typed using the caret character ^. For example, type x^2 as x^2)

The XML parsing instructions are given in Appendix Example 2 (AE2).

Two tests are applied to the student input and they must both return a Boolean true value for the question to be marked correct. The two tests are enclosed in the logical <and> container (AE2 lines 2 and 21). The first test is a regular expression match <regexp> (AE2 lines 4-9), which has been negated by the logical <not> tag (AE2 lines 3 and 10). This checks that the student input does not contain three pairs of parentheses i.e. ( )( )( ) or ( )*( )*( ), which would mean that the student had not done any expansion.

The second test (AE2 lines 11-20) specifies a script to be run by the CAS *Maxima*. The script is specified in the <script> container (AE2 lines 16-18) and expands the difference between the given expression and the prefiltered student input (the @ symbol). The result is compared to the value specified in the <value> element (AE2 line 19) and if it matches, the result of the test is Boolean true. Since the two tests are contained in an

<and> element, the final result is the logical AND of the two tests. Notice that any necessary prefiltering must be done in each test separately, as each test is passed a copy of the student's raw input. Also notice, that this code would allow a partial expansion to be marked correct; if the student has correctly removed only one pair of parentheses, then the answer would be marked correct. Of course, this can be changed, if desired, to award only part marks, or no marks, for a partial expansion.

The capacity to prefilter input allows OTS to handle informal input syntax, which can be defined separately for each answer, if necessary, and the ability to combine different tests together allows sophisticated control over how an input is evaluated. Thus partial credit can be awarded for an answer that is not correct, but contains some of the correct working (Ashton, Beevers, Korabinski & Youngson, 2006; Darrah, Fuller & Miller, 2010).

Modular and self-validating

Using XML to specify both answer format and system configuration means that the OTS configuration is modular and self-validating. Invalid specifications or missing fields can be quickly detected and an invalid specification will not be parsed. Validation is done against a RELAXNG schema, chosen because RELAXNG has the capacity to specify data types, but in a way that is easier to use than other schema languages like XML Schema. A typical question specification is given in Appendix Example 3 (AE3).

Each question must have a <title> container, a <solution> container, which specifies how to determine the correct answer (see AE1 and AE2), and optionally, a <reminder> container, the contents of which are returned when the student selects Check, and lastly <feedback> containers which are returned when Submit is selected and the question is marked. Feedback may differ depending upon whether the question is correct or not. Since title, reminders and feedback are provided via a web page, they must be formatted as valid XHTML.

As an authoring language though, XML is rather daunting for some authors. A future plan is to develop an authoring front end to the OTS which would make it accessible to more users.

Data entry types and structures

So far OTS has developed the following data types and structures: integer and variable-precision floating point numbers; inequalities; regular expressions; an algebraic data type, for passing scripts to the CAS and giving access to all the functionality of *Maxima*; a choice data type, for multiple choice questions; and a general list data type, comprising comma-separated values which can be any of the other data types.

The list data type is very useful for questions such as:

> Solve the following cubic equation for x: $x^3 - 4x^2 + x + 6 = 0$
>
> (Your answers should be a list separated by commas.)

where one does not want to disclose to students that three answers are expected. Lists also give partial credit for some correct answers.

These data types and structures, combined with the suite of prefilter actions discussed earlier, provide fine-grained control over how student input is processed and evaluated. Thus, allowing the informal input syntax to be implemented. For example, equivalences such as $2.5 = 2 + 1/2 = 5/2$ can be handled seamlessly without troubling the student with format directions.

Multistage decision tree

The implementation of the logical combinators <and>, <or> and <not> allows arbitrary combinations of tests to be performed on student input for any question. Since most of the data types are implemented directly in PHP, processing is very efficient and the CAS *Maxima* is only called when necessary.

Server-side processing

On the server, we have opted simply for tab-separated text files to capture the student input and log system activity. Reporting is currently web-based, however, back-end processing could easily be adapted to other formats, like SQL databases, which would allow more sophisticated query and reporting features.

When starting processes remotely via web interfaces, care must be taken to ensure that the server does not become overloaded, either with running processes, or processes that fail to terminate correctly. To ensure that the server performance is not compromised the following control measures were adopted.

- A time limit of 30 seconds was placed on all running PHP scripts. A script cannot run longer than this preset time.
- An adjustable maximum number of simultaneous submissions (initially set at 20) are allowed at any one time.
- External processes, such as *Maxima*, are run via a small program that monitors the executable and will terminate the process if it exceeds a preset time limit. For *Maxima* we have found a time limit of 15 seconds adequate for our purposes. This "monitoring program" is essential as *Maxima* can be locked into an infinite loop when asked to process strange input.

With these measures in place, server bottlenecks have been avoided and the OTS availability has been high.

Implementation and experience so far

The OTS has been used to process entry-level mathematics students since 2006, enabling hundreds of students to be assessed and counseled appropriately. Students find interacting with the system over the web very natural; it can be can be accessed from anywhere in the world, and is available at night and over weekends, for students to use whenever convenient.

Using CAA has resulted in a huge time saving for staff and provides very early feedback to staff and students alike. During initial testing, student feedback was sought via anonymous exit questionnaires, from which a number of modifications were made to the student interface. The main issues identified were to do with question design and errors of logic in the marking scheme. There were no problems with the underlying engine. Subsequently, the OTS was deployed to deliver the first assignment in seven

entry-level mathematics courses, for cohorts in Business, Engineering, Education and Sciences. Student feedback is routinely collected in all courses at USQ via anonymous online questionnaires at the end of each semester. This was one source of information about student reaction to the OTS; the other was comments and questions posted directly by students, either by email or on course online discussion groups, while they were using the system to complete their assignments. We emphasised that fair comment would always be listened to and that electronic marks would be reviewed if requested. With experience, we were better able to anticipate the variations that would arise in student solutions to each question, and to tailor the marking scheme and question feedback appropriately.

The majority of students accept the system well and those who do badly on the test later report that it served as a wake-up call to get them to revise their skills. To date there have been no complaints about accessing and using the system, but some students mistrust the reliability of the marking. Sangwin and Ramsden (2007) report similar fears among students, that they may be marked wrong because of a syntax error. Indeed, some still feel that this was the case even when investigation shows that their error was mathematical. Nonetheless, where a student is still not happy, we either let the student take the test again or mark the test manually, which currently occurs in about 6% of papers.

The main difficulty is developing and testing good questions and specifying the informal syntax for their solution. We learned from student responses and no system can handle perfectly all the possible answers that students may give to any question. Student input was analysed to detect common error patterns and this information was fed back both to question design and solution parsing. This process involved some trial and error, though as we gained more experience using the system, problems became less frequent.

As can be seen from the example questions in this paper, we have tried to help students by giving specific syntax hints where necessary. However, Sangwin and Ramsden (2007) report that around 18% of students still make errors of this kind despite explicit syntax instructions. Their students were expected to enter answers directly in CAS syntax, but in our experience with OTS the incidence of such errors is much lower, around 5%. We believe this is due to the effort we have made to implement an informal input syntax.

Future development

The OTS has shown the feasibility of linking CAA with CAS and implementing an informal linear syntax for student input. As an authoring language though, XML is rather daunting for some authors. It is planned to develop an authoring front-end to the OTS, which would allow questions and answers to be written within one document. The document could be parsed to generate the necessary XML configuration files, the question paper as PDF, and the associated web pages as XHTML. Such a system would make the OTS accessible to many more users.

Currently, the back-end administrative reports and functions use a purpose-built web interface, but capturing the data in an SQL database would integrate better with other data systems at the University. We also plan to investigate more elaborate user verification and access control to enable summative testing online.

We want to develop better-targeted feedback to the student about particular areas of weakness: algebra, functions, trigonometry, inequalities etc and possibly do some syntax checking of student input during the *Check* phase before submission. This would alert the student to possible syntax errors before they are assessed. One possibility is to use *Maxima* to render student input as TEX and pass this to a display engine like *MathJax* (see http://www.mathjax.org/), which could render and display the input as properly formatted mathematics. The student could then be asked to verify, during the Check phase, that the input is what they intended.

The modular design of OTS was intended to allow for automated testing, randomly generating tests from question banks, or targeting tests to areas of weakness. These functions will develop over time and provide web-based assessment, remedial and support materials to students. Another intended improvement is to paginate the *Answer Form* and allow students to save partially completed answers, which will allow for longer tests to be delivered.

Recommendations

We suggest that the following principles may be helpful to designers and developers of computerised assessment systems.

1. Separate question delivery from answer collection. Forcing students to sit at the computer to both read the question and enter the answer in the same session is unnecessarily stressful, especially if the student is unfamiliar with the technology. Allowing the students to work off line using their standard tools reduces this stress.

2. Give instructions about how the answer is to be formatted. Students fear being marked wrong simply because of a formatting error. Hence, the author must think about how the answers will be formatted when writing the question.

3. Develop a robust informal input syntax. In order to move away from fixed format or multiple choice questions, we need to be able to parse algebraic input and numeric input in a variety of forms. Data typing of input helps the different processing required, as does allowing more than one correct answer to a question.

4. Anticipate the variety of input expected on any question. For example, will a number be an integer, fraction or decimal, or possibly all three. If rounding is required, how will this be assessed? Where algebraic input is expected how will it be parsed? This kind of flexibility is invaluable in being able to ask more complex questions. With experience, one can predict the most likely student answers and give credit, or partial credit, to those which are correct.

5. Provide feedback and allow the student to check their answers before final submission. This reduces student anxiety about being marked wrong and increases the value of the test as a learning opportunity.

6. Allow a computer mark to be manually overridden. No automated system is perfect and knowing that there is a right of appeal greatly reduces student anxiety. Our early tests contained multiple errors, but we fixed and refined them over time. As we gained experience, the error rate declined, as did the number of requests for manual marking.

References

Artigue, M. (2002). Learning mathematics in a CAS environment: The genesis of a reflection about instrumentation and the dialectics between technical and conceptual work. *International Journal of Computers for Mathematical Learning*, 7(3), 245-274. http:/dx.doi.org/10.1023/A:1022103903080

Ashton, H. S., Beevers, C. E., Korabinski, A. A. & Youngson, M. A. (2006). Incorporating partial credit in computer-aided assessment of mathematics in secondary education. *British Journal of Educational Technology*, 37(1), 93-119. http:/dx.doi.org/10.1111/j.1467-8535.2005.00512.x

Berger, M. (2010). Using CAS to solve a mathematics task: A deconstruction. *Computers & Education*, 55(1), 320-332. http://dx.doi.org/10.1016/j.compedu.2010.01.018

Darrah, M., Fuller, E. & Miller, D. (2010). A comparative study of partial credit assessment and computer-based testing. In *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2010* (pp. 2335-2340). Toronto, Canada: AACE. http://www.editlib.org/p/34965

Dopper, S. M. & Sjoer, E. (2004). Implementing formative assessment in engineering education: the use of the online assessment system Etude. *European Journal of Engineering Education*, 29(2), 259-266. http://dx.doi.org/10.1080/0304379032000157187

Engelbrecht, J. & Harding, A. (2004). Combining online and paper assessment in a web-based course in undergraduate mathematics. *Journal of Computers in Mathematics and Science Teaching*, 23(3), 217-231. http://www.editlib.org/p/4981

Feng, M., Heffernan, N. & Koedinger, K. (2009). Addressing the assessment challenge with an online system that tutors as it assesses. *User Modeling and User-Adapted Interaction*, 19(3), 243-266. http:/dx.doi.org/10.1007/s11257-009-9063-7

Jourdan, N., Cretchley, P. & Passmore, T. (2007). Secondary-tertiary transition: What mathematics skills can and should we expect this decade? In J. Watson & K. Beswick (Eds), *Mathematics: Essential research, essential practice. Vol. 2 of Proceedings of 30th Annual Conference* (pp. 463-442). Hobart, Australia: MERGA. http://www.merga.net.au/documents/RP412007.pdf

Klai, S., Kolokolnikov, T. & Van den Bergh, N. (2000). Using Maple and the web to grade mathematics tests. In *Proceedings of the international workshop on advanced learning technologies*. 4-6 December, Palmerston North, New Zealand. [verified 29 Aug 2011] http://users.ugent.be/~nvdbergh/aim_rug/docs/iwalt.pdf

Klasa, J. (2010). A few pedagogical designs in linear algebra with Cabri and Maple. *Linear Algebra and its Applications*, 432(8), 2100-2111. http://dx.doi.org/10.1016/j.laa.2009.08.039

Naismith, L. & Sangwin, C. J. (2004). Computer algebra based assessment of mathematics online. In *8th Annual CAA Conference* (pp. 235-242). Loughborough, UK: Loughborough University. http://hdl.handle.net/2134/1956

Nicaud, J.-F., Bouhineau, D. & Chaachoua, H. (2004). Mixing microworld and CAS features in building computer systems that help students learn algebra. *International Journal of Computers for Mathematical Learning*, 9(2), 169-211. http://dx.doi.org/10.1023/B:IJCO.0000040890.20374.37

Sangwin, C. J. (2003). Assessing higher mathematical skills using computer algebra marking through AIM. In R. L. May & W. F. Blyth (Eds), EMAC2003: *Proceedings of the engineering mathematics and applications conference*. University of Technology, Sydney: Engineering Mathematics Group, ANZIAM. http://web.mat.bham.ac.uk/C.J.Sangwin/Publications/Emac03.pdf

Sangwin, C. J. (2006). Assessment within computer algebra rich learning environments. In Le Hung Son, N. Sinclair, J. B. Lagrange & C. Hoyles (Eds.), *Proceedings of the ICMI 17 study conference: Background papers for the ICMI 17 study* (p. C79). Hanoi University of Technology: Springer. http://web.mat.bham.ac.uk/C.J.Sangwin/Publications/2006-ICMI-Sangwin-CJ.pdf

Sangwin, C. J. & Grove, M. J. (2006). STACK: Addressing the needs of the "neglected learners". In M. Seppälä, S. Xambo & O. Caprotti (Eds.), *Proceedings of the WebAlt conference*. Technical University of Eindhoven, Netherlands: European eContent EDC-22253-WEBALT. [verified 29 Aug 2011; 4.53 MB]
http://webalt.math.helsinki.fi/webalt2006/content/e31/e176/webalt2006.pdf

Sangwin, C. J. & Ramsden, P. (2007). Linear syntax for communicating elementary mathematics. *Journal of Symbolic Computation*, 42(9), 920-934.
http://web.mat.bham.ac.uk/C.J.Sangwin/Publications/2007-Sangwin_Ramsden_Syntax.pdf; http://dx.doi.org/10.1016/j.jsc.2007.07.002

Strickland, N. (2002). Alice Interactive Mathematics. *MSOR Connections*, 2(1), 27-30.
http://mathstore.ac.uk/newsletter/feb2002/pdf/aim.pdf

Wang, H.-C., Chang, C.-Y. & Li, T.-Y. (2008). Assessing creative problem-solving with automated text grading. *Computers & Education*, 51(4), 1450-1466.
http://dx.doi.org/10.1016/j.compedu.2008.01.006

Wang, T., Su, X., Wang, Y. & Ma, P. (2007). Semantic similarity-based grading of student programs. *Information and Software Technology*, 49(2), 99-107.
http://dx.doi.org/10.1016/j.infsof.2006.03.001

Wang, T.-H. (2010). Implementation of Web-based dynamic assessment in facilitating junior high school students to learn mathematics. *Computers & Education*, 56(4), 1062-1071.
http://dx.doi.org/10.1016/j.compedu.2010.09.014

Appendix: XML code examples

```
<solution>
    <regexp>
      <prefilter action="remove white space"/>
      <prefilter action="lower case"/>
      <prefilter action="remove stars"/>
      <prefilter action="substitute parenthesis"/>
      <prefilter action="remove brackets"/>
      <value>@^(16b|b16)/a$@</value>
    </regexp>
</solution>
```

Example 1: prefiltering of input allows informal input syntax

```
1  <solution>
2    <and>
3       <not>
4       <regexp>
5         <prefilter action="substitute parenthesis"/>
6         <prefilter action="remove white space"/>
7         <prefilter action="add stars"/>
8         <value>@^\(.+\)\*\(.+\)\*\(.+\)$@</value>
9       </regexp>
10       </not>
11       <maxima>
12         <prefilter action="substitute parenthesis"/>
13         <prefilter action="remove white space"/>
14         <prefilter action="lower case"/>
```

```
15          <prefilter action="add stars"/>
16          <script>
17            display2d:false$ expand((x+1)*(-2*x+1)*(x-3)-(@));
18          </script>
19          <value>0</value>
20        </maxima>
21     </and>
22   </solution>
```

Example 2: multiple tests applied to input allow fine-grained response and partial credit

```
<question id="Q1">
      <title>Question 1</title>
        <solution>
          <float round='1'>
            <prefilter action="remove white space"/>
            <value>-0.0800</value>
          </float>
        </solution>
        <reminder>
          <strong>Remember:</strong> you should supply a negative
           answer.
        </reminder>
        <feedback result="0">
         Unfortunately your answer was incorrect. Did you remember
         to supply a negative answer?
        </feedback>
        <feedback result="1">
         <h3>Congratulations you were correct</h3>
        </feedback>
  </question>
```

Example 3: question specifications are validated using an XML schema

Authors: Mr Tim Passmore, Lecturer, Department of Mathematics and Computing, University of Southern Queensland, Toowoomba Qld 4350, Australia. Email: tim.passmore@usq.edu.au Web: http://www.sci.usq.edu.au/staff/passmore/

Dr Leigh Brookshaw, Lecturer, Department of Mathematics and Computing, University of Southern Queensland, Toowoomba Qld 4350, Australia. Email: leigh.brookshaw@usq.edu.au

Dr Harry Butler, Lecturer, Department of Mathematics and Computing, University of Southern Queensland, Toowoomba Qld 4350, Australia. Email: harry.butler@usq.edu.au Web: http://www.sci.usq.edu.au/staff/butler/

Please cite as: Passmore, T., Brookshaw, L. & Butler, H. (2011). A flexible, extensible online testing system for mathematics. *Australasian Journal of Educational Technology*, 27(6), 896-906. http://www.ascilite.org.au/ajet/ajet27/passmore.html